Sheet 7 (Interactive graphics)

1. In an interactive computer graphics program, the output of the program changes upon a user input. This input could be a mouse click, mouse move, pen flicking or a keyboard input as examples. By design, non-interactive computer graphics program output does not respond to user inputs during its execution.
   To implement an interactive OpenGL computer graphics program, we need to use the undelaying operating system windows and input-output components. The interaction with these components is system dependent. For example, the interaction with the X-window on Unix/Linux operating system is different from that with the Microsoft windows. Hence, a specific window-system API is required for each operating system to fully utilize its capabilities from OpenGL. A basic set of common window-system functionalities are implemented in a cross-platform library called GLUT. An OpenGL programmer can use GLUT library to implement the basic interaction with the underlying windowing and input/output system. GLUT is cross-platform on the function definitions and declaration level, but, of course, is system dependent on the binary-level.

2. Physical input device are the devices employed by the user to input some data to a computer program. They are usually classified according to the type of data they can enter as follows.
   - Text input devices: The keyboard is the primary text input device but other devices exist like character recognition systems and data files/streams
   - Pointing devices: They are used to input location data. The primary example is the standard mouse. Other examples include trackball, data tablets, pens/stylus, joysticks, and space ball. A pointing device may be sub-classified according to the degree of freedom it allows as follows
     - Two degrees of freedom devices like the mouse and trackball
     - Three degrees of freedom like laser scanner that san three dimensional objects and input their three dimensional data points
     - More than three degrees of freedom like the space ball the allow pointing in three independent directions plus twisting in there independent directions

   Logical input devices are the means of inputting data from the application point of view. These are the APIs and data input interface widgets. Some examples are scanf function in the c language, cin object in the C++ language, List boxes and menus in a windowing environment. They can sub-classified according to the nature of the data they provide as follows:
   - String devices: a software object or API that abstract physical text input devices such as cin and scanf
   - Location devices: a software object or API that provide location specified by and physical pointing device. For example, any physical pointing device connected to a windows operating system gives a Mouse Click event when clicked. Here the Mouse is used as a logical pointing device; the click may be originated from a stylus.
   - Pick devices: are used for picking an object from several
   - Choice devices: ere used to select one options from several. List boxes are familiar examples.
   - Valuators devices: are used to input values. Numerical Up/Down controls for changing

values like dates and times are some examples.

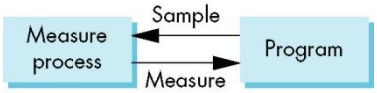
- Stoke devices. Are used to input array of locations are a result of moving a physical input device such as a mouse or a pen.

3. a) The measure process is the process the device does to prepare the data to the applications. For example, the measure process for a keyboard is the process of stoking the input character codes in a keyboard buffer to be ready for the application. The measure process of a mouse is the continuous conversion of locations based on the mouse ball movement. The trigger is an action that determines the start or end of a measure process or the moment at which the current data obtained from the measure process should be handed to the application. For example, the trigger for a text input could be pressing the Enter key. For the mouse, the trigger could be a click.

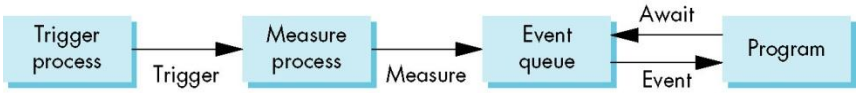
   b) The three distinct modes are
   - Request mode: The application requests input. The measure process starts until the trigger is activated. The input is handed to the application. The keyboard input is an example.
   - Sample mode: The measure process in continuous. When the application need, it can take a sample (read the current measure value). The mouse move then click is an example.
   - Event mode: The trigger causes the input mechanism fire an event in providing the measure values at the triggering moment. It's up to the application program to handle or ignore the event

The following three figure explain the three modes

Request mode



Sample mode



Event mode

4. This can be, and usually is, arranged in a Client-Server relationship. The input data of a given physical input device is available as an input service available to all clients which are the application programs and other components on the system or on another system connected through a network.

5. a) glPushAttrib(GL_ALL_ATTRIB_BITS): This function call makes all the contents of the attribute bits to be pushed on the stack. These attribute bits maintain the current value of the state machine such as the current drawing color and clear color.

   b) glPushMatrix(): this function call makes all the contenets of the transformation and

viewing matrices are pushed on the stack. These matrices specify the current transformation that are applied to the vertices before projected and how they are projected.

c) glPopAttrib(): Reloads the attribute bits, hence the corresponding state machine, from the stack.

d) glPopMatrix(): Reloads the transformation and viewing matrices from the stack

These functions are used when we need to change but keep the current setting of the Gl machine. This is accomplished by pushing the contents of the attributes bits and the matrices on the stage before changing them. We then change these values as needed. When we finish our task with the changed setting we return to the original state by popping the attributes and the matrices. On situation where we need these operations is when calling a display list to execute it. If this display list has to change the current state, we can keep the original state by pushing them at the beginning of the list code and popping it before leaving the list code. In this way the list executes an return without affecting the current state.

6.

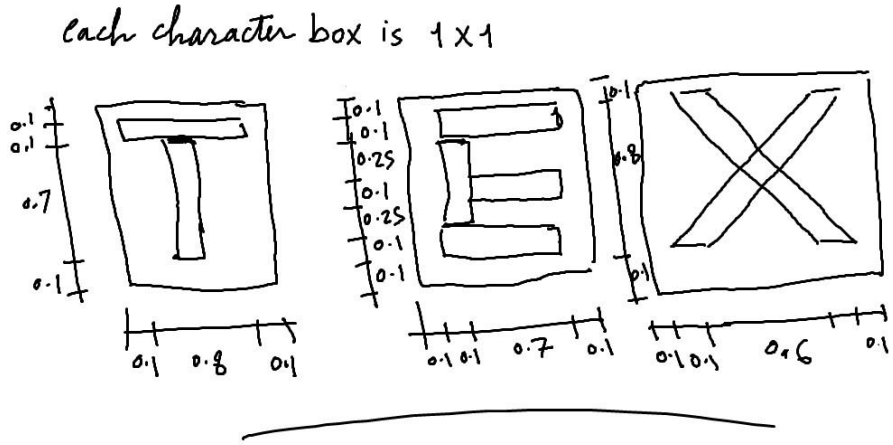
```
#include "stdafx.h"
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    // the string to write
    char *st="Computer Graphics";
    // the starting position of the following text
    // each character drawing prepare to the next
    //by making a translation to it lower right box corner
    glTranslatef(-1000,1000,0);
    for(int i=0;i<17;i++)
    {
        glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN,st[i]);
    }
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(-1000,500,0);
    for(int i=0;i<17;i++)
    {
        glutStrokeCharacter(GLUT_STROKE_ROMAN,st[i]);
    }
    // go to the starting raster position for the following text
    glRasterPos2i(-1200,-500);
    for(int i=0;i<17;i++)
    {
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13,st[i]);
    }
    glFlush();
}

void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-2000.0, 2000.0, -2000.0, 2000.0);
    glMatrixMode(GL_MODELVIEW);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glColor3f(0.0,0.0,0.0);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("GLUT fonts");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

7.


each character box is 1 X 1
0.1
0.1
0.7
0.1
0.1
0.8
0.1
0.1
0.1
0.25
0.1
0.25
0.1
0.1
0.1
0.1
0.7
0.1
0.8
0.1
0.1
0.1
0.6
0.1

```
#include "stdafx.h"
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>
#include <string.h>
GLuint fontBase;

// create the font (the character box is 1 by 1, 0.1 is leaved empty surrounding the character body)
void CreateCharacter(char c)
{
    switch(c)
    {
    case 'T':
        glBegin(GL_QUADS);
        // The horizontal bar
        glVertex2f(0.1,0.9);
        glVertex2f(0.9,0.9);
        glVertex2f(0.9,0.8);
        glVertex2f(0.1,0.8);
        // The vertical bar
        glVertex2f(0.45,0.8);
        glVertex2f(0.55,0.8);
        glVertex2f(0.55,0.1);
        glVertex2f(0.45,0.1);
        glEnd();
        // prepare to the next character
        glTranslatef(1,0,0);
        break;
    case 'E':
        glBegin(GL_QUADS);
        // the upper horizontal bar
        glVertex2f(0.1,0.9);
        glVertex2f(0.9,0.9);
        glVertex2f(0.9,0.8);
        glVertex2f(0.1,0.8);
        // the vertical bar
        glVertex2f(0.1,0.8);
        glVertex2f(0.2,0.8);
        glVertex2f(0.2,0.2);
        glVertex2f(0.1,0.2);
        // The middle horozontal bar
        glVertex2f(0.2,0.45);
        glVertex2f(0.9,0.45);
        glVertex2f(0.9,0.55);
        glVertex2f(0.2,0.55);
        // the lower horizontal bar
        glVertex2f(0.1,0.2);
        glVertex2f(0.9,0.2);
        glVertex2f(0.9,0.1);
        glVertex2f(0.1,0.1);
        glEnd();
        // prepare to the next character
        glTranslatef(1,0,0);
        break;
    case 'X':
        glBegin(GL_QUADS);
        // the left to right bar
        glVertex2f(0.1,0.9);
        glVertex2f(0.2,0.9);
        glVertex2f(0.9,0.1);
        glVertex2f(0.8,0.1);
        // the right to left bar
        glVertex2f(0.8,0.9);
        glVertex2f(0.9,0.9);
        glVertex2f(0.2,0.1);
        glVertex2f(0.1,0.1);
```

```
67          glEnd();
68          // prepare to the next character
69          glTranslatef(1,0,0);
70          break;
71      default:
72          // draw square for not implemented characters
73          glBegin(GL_QUADS);
74          // the square
75          glVertex2f(0.1,0.9);
76          glVertex2f(0.9,0.9);
77          glVertex2f(0.9,0.1);
78          glVertex2f(0.1,0.1);
79          glEnd();
80          // prepare to the next character
81          glTranslatef(1,0,0);
82          break;
83      }
84  }
85  void CreateFont()
86  {
87      // return the index first of 256 consecutive avilable ids
88      fontBase=glGenLists(256);
89      for(int i=0;i<256;i++)
90      {
91          glNewList(fontBase+i,GL_COMPILE);
92          CreateCharacter(i);
93          glEndList();
94      }
95  }
96
97  void display()
98  {
99      glClear(GL_COLOR_BUFFER_BIT);
100     // the string to write
101     char *st="TEXT";
102     // the starting position of the following text
103     // each character drawing prepare to the next
104     //by making a translation to it lower right box corner
105     glTranslatef(-3,2,0);
106     glListBase(fontBase);
107     glCallLists((GLint) strlen(st), GL_BYTE, st);
108     glFlush();
109 }
110
111 void myinit()
112 {
113     glMatrixMode(GL_PROJECTION);
114     glLoadIdentity();
115     gluOrtho2D(-5.0, 5, -5, 5);
116     glMatrixMode(GL_MODELVIEW);
117     glClearColor (1.0, 1.0, 1.0, 1.0);
118     glColor3f(0.0,0.0,0.0);
119 }
120
121 int main(int argc, char **argv)
122 {
123     glutInit(&argc, argv);
124     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
125     glutInitWindowSize(500, 500);
126     glutCreateWindow("Font creation");
127     glutDisplayFunc(display);
128     myinit();
129     CreateFont();
130     glutMainLoop();
131 }
```